



Servidor Web

É quando você finalmente entende por que um URL começa com HTTP

Resumo:

Este projeto trata de escrever seu próprio servidor HTTP.

Você poderá testá-lo com um navegador real.

HTTP é um dos protocolos mais usados na internet.

Conhecer seu mistério será útil, mesmo se você não estiver trabalhando em um site.

Versão: 21.2

Conteúdo

EU	Introdução	2
II	Regras gerais	3
III	Parte obrigatória	4
III.1	Requisitos	6
III.2	Apenas para MacOS.	7
III.3	Arquivo de configuração	7
IV	parte bônus	9
V	Submissão e avaliação por pares	10

Capítulo I

Introdução

O **Hypertext Transfer Protocol** (HTTP) é um protocolo de aplicação para sistemas de informação hipermídia distribuídos e colaborativos.

HTTP é a base da comunicação de dados para a World Wide Web, onde documentos de hipertexto incluem hiperlinks para outros recursos que o usuário pode acessar facilmente. Por exemplo, com um clique do mouse ou tocando na tela em um navegador da web.

O HTTP foi desenvolvido para facilitar o hipertexto e a World Wide Web.

A principal função de um servidor web é armazenar, processar e entregar páginas web aos clientes. A comunicação entre cliente e servidor ocorre através do Protocolo de Transferência de Hipertexto (HTTP).

As páginas entregues são mais frequentemente documentos HTML, que podem incluir imagens, folhas de estilo e scripts, além do conteúdo do texto.

Vários servidores web podem ser usados para um site de alto tráfego.

Um agente de usuário, geralmente um navegador ou rastreador da Web, inicia a comunicação solicitando um recurso específico usando HTTP e o servidor responde com o conteúdo desse recurso ou com uma mensagem de erro se não for possível fazê-lo. O recurso normalmente é um arquivo real no armazenamento secundário do servidor, mas isso não é necessariamente o caso e depende de como o servidor web é implementado.

Embora a função principal seja servir conteúdo, a implementação completa do HTTP também inclui formas de receber conteúdo dos clientes. Este recurso é utilizado para envio de formulários web, incluindo upload de arquivos.

Capítulo II

Regras gerais

- Seu programa não deve travar em nenhuma circunstância (mesmo quando ficar sem memória) e não deve encerrar inesperadamente.
Caso isso aconteça, seu projeto será considerado não funcional e sua nota será 0.
- Você terá que entregar um Makefile que irá compilar seus arquivos fonte. Não deve vincular novamente.
- Seu Makefile deve conter pelo menos as regras:
\$(NOME), tudo, limpo, fclean e re.
- Compile seu código com c++ e os sinalizadores -Wall -Wextra -Werror
- Seu código deve estar em conformidade com o **padrão C++ 98**. Então, ele ainda deve compilar se você adicionar o sinalizador -std=c++98.
- Tente sempre desenvolver usando o máximo de recursos C++ que puder (por exemplo, escolha <cstring> em vez de <string.h>). Você tem permissão para usar funções C, mas sempre prefira suas versões C++, se possível.
- Qualquer biblioteca externa e bibliotecas Boost são proibidas.

Capítulo III

Parte obrigatória

Nome do programa	servidor web
Entregar arquivos	Makefile, *.{h, hpp}, *.cpp, *.tpp, *.ipp, arquivos de configuração
Makefile	NOME, tudo, limpo, fclean, re
Argumentos	[Um arquivo de configuração]
Funções externas.	Tudo em C++ 98. execve, dup, dup2, pipe, strerror, gai_strerror, errno, dup, dup2, fork, socketpair, htons, htonl, ntohs, ntohl, seleccione, enquete, epoll (epoll_create, epoll_ctl, epoll_wait), kqueue (kqueue, kevent), soquete, aceitar, ouvir, enviar, recv, chdir bind, conectar, getaddrinfo, freeaddrinfo, setsockopt, getsockname, getprotobyname, fcntl, fechar, ler, escrever, waitpid, matar, sinalizar, acessar, stat, abrir, open, readdir e closedir.
Libft autorizado	n / D
Descrição	Um servidor HTTP em C++ 98

Você deve escrever um servidor HTTP em C++ 98.

Seu executável será executado da seguinte forma:

`./webserv [arquivo de configuração]`



Mesmo que poll() seja mencionado no assunto e na escala de avaliação, você pode usar qualquer equivalente, como select(), kqueue() ou epoll().



Por favor, leia o RFC e faça alguns testes com telnet e NGINX antes iniciando este projeto.

Mesmo que você não precise implementar toda a RFC, lê-la o ajudará a desenvolver os recursos necessários.

III.1 Requisitos

- Seu programa precisa usar um arquivo de configuração como argumento ou usar um caminho padrão.
- Você não pode executar outro servidor web.
- Seu servidor nunca deve bloquear e o cliente pode ser devolvido corretamente, se necessário.
- Deve ser não bloqueador e usar apenas 1 poll() (ou equivalente) para todas as operações de I/O entre o cliente e o servidor (escuta incluída).
- poll() (ou equivalente) deve verificar a leitura e a gravação ao mesmo tempo.
- Você nunca deve fazer uma operação de leitura ou gravação sem passar por poll() (ou equivalente).
- A verificação do valor de erro é estritamente proibida após uma operação de leitura ou escrita.
- Você não precisa usar poll() (ou equivalente) antes de ler seu arquivo de configuração.



Como você precisa usar descritores de arquivo sem bloqueio, é possível usar funções de leitura/recv ou gravação/envio sem poll() (ou equivalente), e seu servidor não estaria bloqueando.

Mas consumiria mais recursos do sistema.

Assim, se você tentar ler/receber ou escrever/enviar qualquer descritor de arquivo sem usar poll() (ou equivalente), sua nota será 0.

- Você pode usar qualquer macro e definir como FD_SET, FD_CLR, FD_ISSET, FD_ZERO (entender o que e como eles fazem isso é muito útil).
- Uma solicitação ao seu servidor nunca deve ficar suspensa para sempre.
- Seu servidor deve ser compatível com o **navegador** de sua escolha.
- Consideraremos que o NGINX é compatível com HTTP 1.1 e pode ser usado para comparar cabeçalhos e comportamentos de resposta.
- Seus códigos de status de resposta HTTP devem ser precisos.
- Seu servidor deverá ter **páginas de erro padrão** se nenhuma for fornecida.
- Você não pode usar fork para algo diferente de CGI (como PHP, ou Python, e assim por diante).
- Você deve ser capaz de **servir um site totalmente estático**.
- Os clientes devem ser capazes de **fazer upload de arquivos**.
- Você precisa de pelo menos os métodos GET, POST e DELETE.
- Testes de estresse em seu servidor. Deve permanecer disponível a todo custo.
- Seu servidor deve ser capaz de escutar múltiplas portas (consulte *Arquivo de configuração*).

III.2 Apenas para MacOS



Como o MacOS não implementa `write()` da mesma forma que outros sistemas operacionais Unix, você pode usar `fcntl()`.

Você deve usar descritores de arquivo em modo sem bloqueio para obter um comportamento semelhante ao de outros sistemas operacionais Unix.



No entanto, você só pode usar `fcntl()` com os seguintes flags: `F_SETFL`, `O_NONBLOCK` e

`FD_CLOEXEC`.

Qualquer outra bandeira é proibida.

III.3 Arquivo de configuração



Você pode se inspirar na parte 'servidor' do arquivo de configuração NGINX.

No arquivo de configuração, você deverá ser capaz de:

- Escolha a porta e o host de cada 'servidor'.
- Configure os `server_names` ou não.
- O primeiro servidor para um `host:port` será o padrão para este `host:port` (isso significa que ele responderá a todas as solicitações que não pertencem a outro servidor).
- Configurar páginas de erro padrão.
- Limite o tamanho do corpo do cliente.
- Configure rotas com uma ou várias das seguintes regras/configuração (as rotas não serão estar usando `regexp`):
 - Defina uma lista de métodos HTTP aceitos para a rota.
 - Defina um redirecionamento HTTP.
 - Defina um diretório ou arquivo de onde o arquivo deve ser pesquisado (por exemplo, se url `/kapouet` tiver raiz em `/tmp/www`, url `/kapouet/pouic/toto/pouet` for `/tmp/www/pouic/toto/pouet`).
 - Ative ou desative a listagem de diretórios.

- Defina um arquivo padrão para responder se a solicitação for um diretório.
 - Execute CGI com base em determinada extensão de arquivo (por exemplo .php).
 - Faça funcionar com métodos POST e GET.
 - Torne a rota capaz de aceitar arquivos carregados e configure onde eles devem ser salvo.
 - Você quer saber o que é um CGI? •
- Como você não chamará o CGI diretamente, use o caminho completo como PATH_INFO.
- Apenas lembre-se que, para solicitações em pedaços, seu servidor precisa descompactá-las, o CGI esperará EOF como final do corpo. • A mesma coisa para a saída do CGI. Se nenhum content_length for retornado do CGI, EOF marcará o final dos dados retornados.
 - Seu programa deverá chamar o CGI com o arquivo solicitado como primeiro argumento.
 - O CGI deve ser executado no diretório correto para acesso ao arquivo de caminho relativo. • Seu servidor deve funcionar com um CGI (php-CGI, Python e assim por diante).

Você deve fornecer alguns arquivos de configuração e arquivos básicos padrão para testar e demonstrar que cada recurso funciona durante a avaliação.



Se você tiver alguma dúvida sobre um comportamento, compare o comportamento do seu programa com o do NGINX. Por exemplo, verifique como funciona server_name. Compartilhamos com você um pequeno testador. Não é obrigatório passar se tudo funcionar bem com seu navegador e testes, mas pode ajudá-lo a encontrar alguns bugs.



O importante é a resiliência. Seu servidor nunca deve morrer.



Não teste com apenas um programa. Escreva seus testes com uma linguagem mais conveniente, como Python ou Golang, e assim por diante. Mesmo em C ou C++, se você quiser.

Capítulo IV

Parte bônus

Aqui estão os recursos extras que você pode adicionar:

- Suporte a cookies e gerenciamento de sessões (preparar exemplos rápidos).
- Lidar com vários CGI.



A parte bônus só será avaliada se a parte obrigatória for PERFEITA. Perfeito significa que a parte obrigatória foi feita integralmente e funciona sem mau funcionamento. Se você não passou em TODOS os requisitos obrigatórios, sua parte do bônus não será avaliada de forma alguma.

Capítulo V

Envio e avaliação por pares

Entregue sua tarefa em seu repositório Git normalmente. Somente o trabalho dentro do seu repositório será avaliado durante a defesa. Não hesite em verificar os nomes dos seus arquivos para garantir que estão corretos.



16D85ACC441674FBA2DF65190663F42A3832CEA21E024516795E1223BBA77916734D1
26120A16827E1B16612137E59ECD492E46EAB67D109B142D49054A7C281404901890F
619D682524F5